DOCUMENT RESUME

ED 388 249 IR 017 398

AUTHOR Guralnick, David; Kass, Alex
TITLE An Authoring System for Creating Computer-Based Role-Performance Trainers.
PUB DATE 94
NOTE 7p.; In: Educational Multimedia and Hypermedia, 1994. Proceedings of ED-MEDIA 94--World Conference on Educational Multimedia and Hypermedia (Vancouver, British Columbia, Canada, June 25-30, 1994); see IR 017 359.
PUB TYPE Reports - Descriptive (141) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Authoring Aids (Programming); Computer Assisted Instruction; *Computer Software Development; *Courseware; Multimedia Materials; Programming; *Training Methods

ABSTRACT

This paper describes a multimedia authoring system called MOPed-II. Like other authoring systems, MOPed-II reduces the time and expense of producing end-user applications by eliminating much of the programming effort they require. However, MOPed-II reflects an approach to authoring tools for educational multimedia which is different from most tools available on the commercial market, the theory-rich authoring tool. General purpose tools, particularly graphical user interface (GUI) tools, force software developers to attempt to map the conceptual components of their well-defined task to the physical components of a computer interface. This effectively takes the target audience of authoring tools, experts in teaching and in specific content areas, and forces them to act as novice computer programmers. The way to allow non-programmers to build good educational software is to give them the tools and let them build software out of constructs they're familiar with. Role-performance trainers involve learning a procedure in the performance of tasks in which there is a fairly regular routine. Typical role-performance trainers may have the following components: simulation; failure identification; scaffolding; many scenarios; Socratic teaching; and modeling. MOPed's theory-rich structure guides non-programmers in building high quality role-performance trainers. A sample application is described, in which MOPed-II teaches employees of a fast-food restaurant to ring up customer orders. A scenario developed may either select a task-structure class from an on-screen palette, or specialize a task structure. (Contains 10 references.) (MAS)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED 388 249

An Authoring System for Creating Computer-Based Role-Performance Trainers

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
☐ This document has been reproduced as received from the person or organization originating it.
☐ Minor changes have been made to improve reproduction quality.
• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

DAVID GURALNICK and ALEX KASS

The Institute for the Learning Sciences
Northwestern University
Evanston, Illinois 60201

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Gary H. Marks
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

Abstract: In this paper we will describe a multimedia authoring system called MOPed-II[1]. Like other authoring systems, MOPed-II greatly reduces the time and expense of producing end-user applications by eliminating much of the programming effort they require. However, MOPed-II reflects an approach to authoring tools for educational multimedia which is rather different from most tools available on the commercial market. In this paper we argue in favor of this approach, which we call theory-rich authoring tools, and describe the success we have had with MOPed-II.

1. Introduction

The most common approach to authoring tools for multimedia applications is the general-purpose construction set, as exemplified by such commercial software programs as HyperCard, SuperCard, ToolBook, and Authorware. These tools are useful for building interface prototypes and simple applications. When used to build educational and training software, however, such tools leave it to individual tool users (that is, software developers) to structure the student's educational experience. The idea of using general-purpose (or theory-neutral) tools to build complex learning environments is predicated on the belief that tool users will be able to produce high-quality educational software if only they can be relieved of the programming burden. We believe that the task of designing as well as programming pedagogically effective software is a difficult challenge. Production of high-quality educational software requires more than technical achievement; it requires insight into what makes for an effective learning experience. Therefore, authoring tools for educational software should help developers design and build effective learning environments. We claim that theory-rich authoring tools which guide a developer in building a learning environment will help tool users build pedagogically sound software quickly and easily. In this paper we justify this claim and describe MOPed-II, a theory-rich authoring tool.

2. Why theory-rich authoring systems are needed

2.1 The importance of authoring tools

Computers have the potential to make great contributions to education and training, but that potential is difficult to realize. At its best, education and training software can, among other things, allow students to work on projects that would be infeasible without the help of computer simulation; provide individual feedback to the student at just the point when it is needed; and make a broad array of expertise and reference material easily accessible. Little such software is actually in use in the field, largely because of the size of the required software-development effort; an enormous amount of software will be required in order for the computer to make a noticeable impact, and high-quality software is expensive to produce. Thus there is an imposing economic barrier blocking the path from the laboratory to the classroom. Innovators may produce a few great pieces of software, but turning a few innovations into enough quality hours of instruction to account for a significant fraction of a curriculum remains a challenge. The problem is made particularly acute by the fact that high-quality educational software often requires technically challenging components such as complex interfaces, multimedia, and simulations.

[1] MOPed-II is based on an earlier authoring system called MOPed, developed by Enio Ohmaye (1992).

IR017398

BEST COPY AVAILABLE

2.2 Theory-rich, not theory-neutral tools

We believe that a crucial step toward realizing the advantages that computer-based training can provide is to increase the number of people that can produce high-quality training software, via the development of authoring systems. But what form should those tools take? The tools currently available come in a variety of forms. Commercial drill-and-practice software tools can make it easy to construct a very narrow range of rigidly-structured learning environments, but those applications tend to be structured according to what is easy to implement on a computer rather than what makes for effective learning. Graphical user-interface tools are increasingly common; they provide the software developer with capabilities such as simple ways to put in graphics and buttons. But GUI tools solve only a portion of the software-building problem, and cause some other problems themselves. Johnson et al. (1993) describe several such problems, including that GUI tools require too much specialized programming skill and often enforce arbitrary forms of consistency (e.g., from Johnson et al., "everything is controlled via menus.") At their best, general-purpose tools are experts at managing computer objects. Educational software developers need tools which help manage pedagogy as well.

The goal of authoring tools for educational software should be to facilitate a clear conceptual mapping from the pedagogical objectives that an author wishes to achieve to an appropriate computer-based learning environment. General-purpose tools, particularly GUI tools, force software developers to attempt to map the conceptual components of their well-defined task to the physical components of a computer interface. This effectively takes the target audience of authoring tools, experts in teaching and in specific content areas, and forces them to act as novice computer programmers. The way to allow non-programmers to build good educational software is to give them tools that let them build software out of constructs they're familiar with, not to make them into pseudo-programmers. Authoring tools for educational and training software should take the form of special-purpose tools geared toward building specific classes of learning environments.

3. What types of theory-rich tools should we build?

3.1 Classifying educational software

At the Institute for the Learning Sciences, we have built a wide range of different types of educational software programs (brief descriptions of some of the Institute's software can be found in [Schank, 1991]), including systems for children (K-12) as well as for professional and military training. The tasks students must perform in our software are also quite varied. For example, some programs allow a student to create something new (e.g., CreAnimate, a biology tutor which lets students create their own animal–see Edelson, 1992), others let a student explore some subject area (e.g., Road Trip, a U.S. geography program which lets students simulate traveling across the country to reach destinations they've chosen–see Kass & Guralnick, 1991; Kass & McGee, 1992); still others have a more-specific task for the student, such as learning a foreign language (e.g., Dustin, a foreign-language tutor aimed at consultants who come to the United States for a training course–see Ohmaye, 1992).

Through our experience in building educational software, several classes of tasks have emerged as common–e.g., exploration tasks, artifact-construction tasks and "role-performance" tasks (defined below). In addition, pieces of software which teach different tasks within a particular class often share similarities in structure. We claim that the best way to construct authoring tools that encourage pedagogically appropriate design is to give them knowledge about task structures. We can then build theory-rich authoring tools for each structural class of educational software that we can identify. The tools' built-in knowledge about task structures helps them provide task-building guidance for software developers. In the remainder of this paper we will define one class of tasks and describe a tool we have built to build learning environments for tasks of that class.

3.2 Role-performance trainers: An example of educational software structure

We define role-performance tasks as tasks in which there is a fairly regular routine, such as operating a piece of equipment. At each point in a role-performance task, there is a correct next step (or small set of acceptable next steps) that the student should take. For example, the real-life task of working as a fast-food restaurant cashier and ringing up customers' orders on a customized cash register is a role-performance task. For each food item in a customer's order, there's a certain key sequence (or possibly a few options) for that item. A good order-ringer knows which keys correspond to which food items and can ring up orders quickly and accurately. Creative problem-solving is not the core job skill; understanding the procedure is. Role-performance tasks can come in large and more complex forms as well. The task of fielding customer complaint phone calls for a large water utility also has a regular routine, though it's harder than ringing up food orders. A customer service representative needs to learn which questions to ask a customer in order to diagnose the cause of the customer's water problem and recommend treatment. A good service representative understands an accepted set of common water problems, causes, and questions to ask a customer. So role-performance tasks involve learning a procedure of some sort, though it may have variations and options, and may be big.

Unfortunately, the procedural nature of role-performance tasks often causes people to try to teach them by rote. For example, one could try to teach cash-register skills by asking a student to memorize sequences of register buttons. Brown, Collins and Duguid's work on situated cognition (1989) showed that the activity during which knowledge is developed is an integral part of what is learned. Rote memorization separates knowledge from the real-life activity it is intended for. Instead, role-performance tasks are best taught by having the student practice the task and learn any required principles in context. Good role-performance trainers (RPTs) enable students to learn the task they are being trained to perform in an environment like the one in which they will actually perform it. Further, good RPTs guide students to understand how to perform the task.

3.3 Components of role-performance trainers

Building an authoring tool for a task class requires an analysis of what components the trainers for that class should have. Role-performance training breaks down into two primary components: practice and guidance. A good role-performance trainer includes a realistic simulation of the actual task for a student to practice on, along with tutoring to help the student understand his successes and (especially) his failures. The class of role-performance tasks includes two subcategories: theory-based role-performance tasks and convention-based role-performance tasks. All role-performance tasks represent procedures. In theory-based role-performance tasks, the correct steps in the task have some non-arbitrary reason for their correctness. A student who understands the underlying theory will be more successful at performing the task. For example, a water utility customer service representative who understands the causal model of how water reaches a customer's home is well-equipped to diagnose customers' water problems. In convention-based tasks, there is no deep theory behind how to perform the task. For example, a student need not go through a complex reasoning process in order to understand the positioning of buttons on a fast-food cash register. So a theory-based role-performance trainer might benefit from some components that a convention-based training system wouldn't need.

Role-performance trainers typically include the following components:

- **Simulation:** Since role-performance trainers allow students to learn by doing–that is, by practicing the task–a good RPT needs a realistic practice environment.
- **Failure Identification:** A role-performance trainer must guide the student in performing the task. When a student makes a mistake, the RPT needs to be able to explain the nature of the failure to the student as well as teach the student how to recover from the failure.
- **Scaffolding:** A role-performance training system must provide support, or *scaffolding* (Collins, Brown, & Newman, 1989), for a student who is performing a task. Scaffolding in a training system essentially means that the system performs or helps the student perform parts of the task that the student cannot.

Certain role-performance trainers also need the following additional components:

- **Many scenarios:** Most role-performance tasks have lots of variations. A guided practice environment for such a task is only useful if it provides lots of different scenarios for a student to practice on.
- **Socratic Tutoring:** Socratic dialogues help students understand the reasoning behind the steps of a procedure (Collins & Stevens, 1983). Training systems for theory-based role-performance tasks can especially benefit from a Socratic tutoring component.
- **Modeling:** It is often helpful for a student to learn how to perform a task in part by watching an expert do it (Collins, Brown, & Newman, 1989). This method is useful mainly for social types of tasks, such as speaking a foreign language (e.g., Ohmaye's [1992] foreign-language trainer, Dustin).

4. MOPed-II: An authoring system for role-performance trainers

MOPed-II is a theory-rich graphical authoring tool for building role-performance training systems. A software developer constructs a MOPed-II application out of conceptual components which represent tasks and subtasks that a student must perform. This is in contrast to general-purpose tools, whose program components are typically physical objects such as buttons or graphics. Rather than force software developers to try to convert content and teaching knowledge into a vocabulary of buttons and programming statements, MOPed-II lets them build the actual program using a vocabulary of task structures. MOPed-II's theory-rich structure guides non-programmers in building high-quality role-performance trainers quickly and easily.

4.1 A sample application built with MOPed-II

In order to understand how MOPed-II aids in build role-performance trainers, it is useful to consider an example. We have built a range of programs with MOPed-II, including a program which teaches foreign-born consultants how to speak English, and an application which teaches customer service representatives from a water utility how to diagnose and treat customers' water problems. Another program which we have built with MOPed-II teaches employees of a fast-food franchise to ring up customers' orders using the restaurant's special-purpose cash register. At the chain for whom this system was built, the job description of "cashier" includes only

events which transpire after the customer's food has been collected and packaged. The cashier's job is essentially to ring up the sale on the cash register and collect money from the customer.

A large portion of the screen in this sample application is taken up with a simulation of the cash register. Both the keyboard and register screen look and act like the real-life register. The student can take any of three different types of actions at any point in a session with the program:

- press a key on the register
- say something to a simulated customer (e.g., "That will be $8.75, please.")
- ask the on-line tutor a question

One area of the screen displays a picture of the food order along with (sometimes) a text description of the items in the order. Another screen region holds two performance meters ("speed" and "accuracy"), which provide the student with feedback. Other sections provide buttons for the student to talk to the customer or ask the tutor a question.

At the beginning of each scenario the student sees a video, in which the "server" (the employee whose job is to package the customer's order) tells the student what the customer has ordered, just as the servers tell the cashiers this information in real life. Figure 1 shows a portion of a transcript from a typical student interaction with the cashier role-performance trainer:

Server (in video): The customer has ordered a half chicken and a small drink.
Student: Presses "Now What?" button
Tutor: You need to ring up the half chicken and ring up the drink.
Student: Presses "How Do I Do That?" button
Tutor: To ring up the half chicken, you need to press the "HALF CHICKEN" key. It is the key you see flashing on the register now. *[key flashes on screen]*
Student: Presses HALF CHICKEN key *[register screen shows half chicken rung up]*
Student: Presses $1.19 DRINK key *[register screen shows $1.19 drink rung up]*
Tutor *[intervenes]*: You don't need to ring up the large drink.
Student: Presses "Why?" button
Tutor: You don't need to ring up the large drink because it's not the right size.
Student: Presses "Now What?" button
Tutor: You need to void out the last item.
Student: Presses "How Do I Do That?" button
Tutor: To void out the last item, you need to press the VOID key twice. It is the key you see flashing on the register now. *[key flashes on screen]*

Figure 1. Sample interaction with the cashier-training application.

The cashier training system runs students through a series of scenarios, each scenario including a different customer order for the student to ring up.

4.2 The structure of MOPed-II applications

Computer-based role-performance trainers can be decomposed into three layers: the interface layer (objects on the screen that the student sees and manipulates), the task environment layer (the engine processing a student's actions, including the simulation component), and the tutoring layer (including hints, suggestions, explanations, demonstrations). MOPed-II supports building of the task environment and tutoring layers along with communication with an interface layer.

A scenario is a series of tasks that a student must perform. Each MOPed-II training application is composed of a number of scenarios. Typically, each scenario is a variation on the standard task of an application. For example, in the cashier training system, the overall task is to ring up customers' orders, and each scenario involves a different customer. Scenarios are composed of subtasks. The main subtasks for the cashier application are ringing up the items on the cash register, collecting money from the customer, ringing up the money correctly and giving the customer change (if any). The details of how and when a student is asked to perform each task in a scenario may vary depending on the student's performance; for example, a student who has not rung up the items correctly cannot move on to the money-collection task.

4.3 Creating scenarios

To create a scenario, an author using MOPed-II describes the structure of the task associated with the scenario by selecting a task-structure class from an on-screen palette. MOPed-II includes a set of predefined general task

structures, called templates, which are common to role-performance tasks. One such structure is the "Unordered Steps" task template, which is used for tasks in which the student must perform certain steps, but not in any particular sequence (e.g., ring up the half chicken and ring up the drink in the example above). Each template provides a simple English-language form which lets the software developer fill out such information as the names of the steps in the task, English descriptions of the steps (for use by the tutor), and reasons why certain steps are correct or incorrect within the context of the particular task. Filling out this form yields a graphical task structure.

Each graphical task structure is composed of a sequence of actions. These actions come in four varieties:

- **output processing**: the system initiates some action upon their execution, such as show a video
- **input processing:** wait for input from the student before continuing processing
- **flow-of-control:** e.g., begin or end a task, branch based on some criteria
- **subtask:** move into another task or subtask

A developer need not utilize a template-generated task structure as-is. It is common for a developer to specialize the general predefined task structure for his own needs–e.g., providing special interface functionality. Figure 2 shows a typical task structure from the cashier training course described above. This structure represents the task from the example above in which the student's goal is to ring up the half chicken and ring up the drink. When a student has achieved that composite goal, the program goes on to the next task. If the student takes a wrong step, tutoring is provided (via a linked program called the Teaching Executive, written by Kemi Jona–see [Jona & Kass, 1993]), and the program enters the appropriate task for recovering from the error (e.g., the "void last item" task).

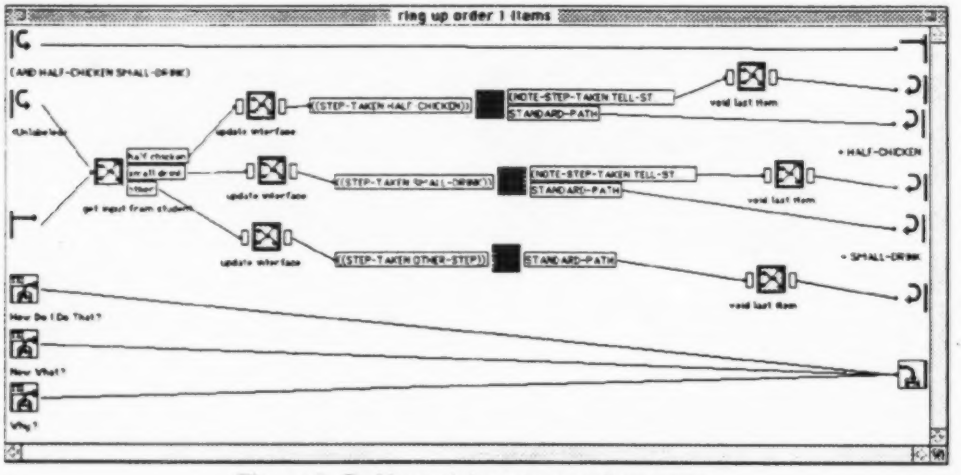

Figure 2. Cashier training course task structure.

So the general MOPed-II task-building process takes the following form: for each task, the developer chooses a general task structure template, fills out its English form, and (perhaps) specializes the task structure a bit more. Occasionally, MOPed-II applications might require a task structure which doesn't have a template. But building tasks from scratch is fairly simple, particularly when template-generated structures provide examples.

4.4 Generating tutoring

As mentioned above, tutoring in MOPed-II is controlled by a linked program called the Teaching Executive. The Teaching Executive provides an architecture for providing tutoring. Tutoring comes in two major forms: remediation (which requires failure identification)–telling a student he made a mistake and helping him understand why (possibly including Socratic tutoring); and support (scaffolding)–giving a student hints about what to do next, or demonstrating a step (possibly including modeling). Remediation and support can range from simple to complex, depending on the particulars of the task and steps the student needs help with. For example, a student who can't find the "Small Drink" key on a cash register simply needs to be shown where it is located, while a student who misdiagnoses a customer's water problem could benefit from a lengthier Socratic dialogue about the basis for his mistaken hypothesis. Most of the work needed to generate simple tutoring in a MOPed-II application is included in the task-building process. MOPed-II tasks include special structures which communicate with the Teaching Executive. The Executive itself houses modules which are called in a particular tutoring situation–e.g., the "Now What?" module, which is called when a student presses the "Now What?" button. More-complex tutoring forms such as Socratic dialogues are handled by separate, special-purpose

BEST COPY AVAILABLE

modules which are invoked by the Teaching Executive at the appropriate times. As with task structures, a set of general Teaching Executive modules is provided. The application-builder chooses the modules he needs and fills in a small amount of application-specific information.

5. Conclusion

In order for computers to have a significant impact on education and training, people need to produce a large quantity of high-quality software. One way to generate lots of software is to expand the group of people who can build it. We can do this by building tools to allow people other than computer programmers to build software. In particular, we need tools for content experts and teachers to use. General-purpose authoring tools such as GUI tools force content experts to map their knowledge of the task to a set of computer interface objects–this turns out to be almost difficult as programming.

Instead, we need a set of theory-rich tools which are built especially for educational and training software, and guide software developers in building pedagogically sound learning environments. We have built MOPed-II, a theory-rich authoring system for role-performance trainers. We believe that theory-rich authoring tools are the best way to help non-programmers build high-quality educational and training software.

6. References

Brown, J.S., Collins, A., & Duguid, P. (1989). Situated cognition and the culture of learning. Educational Researcher, 18, 32-43.

Collins, A., Brown, J.S., & Newman, S.E. (1989). Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. In Resnick, L.B. (Ed.), *Knowing, learning, and instruction: Essays in honor of Robert Glaser* (pp. 453-494). Hillsdale, NJ: Lawrence Erlbaum Associates.

Collins, A. & Stevens, A. (1983). A cognitive theory of inquiry teaching. In Reigeluth, C.M. (Ed.), *Instructional-design theories and models: an overview of their current status*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Edelson, D. (1992) Learning From Stories: Indexing and Reminding in a Socratic Case-Based Teaching System for Elementary School Biology. T.R. 43, Institute for the Learning Sciences, Northwestern University.

Johnson, J.A., Nardi, B.A., Zarmer, C.L., & Miller, J.R. (1993). ACE: Building Interactive Graphical Applications. *Communications of the ACM, 36*, 41-55.

Jona, M., & Kass, A. (1993). The Teaching Executive: Facilitating Development of Educational Software through the Reuse of Teaching Knowledge." In Estes, N. & Thomas, M. (Eds.), *Proceedings of the Tenth International Conference on Technology and Education* (pp. 640-642). Cambridge, MA: Massachusetts Institute of Technology.

Kass, A., & Guralnick, D. (1991). Environments for Incidental Learning: Taking Road Trips Instead of Memorizing State Capitals. In L. Birnbaum (Ed.), *Proceedings of the International Conference on the Learning Sciences* (pp. 258-264). Evanston, IL: Assoc. for the Advancement of Computing in Education.

Kass, A., & McGee, S. (1992). The Road Trip Project: Learning Geography through Simulated Travel. Technical Report 42, Institute for the Learning Sciences, Northwestern University.

Ohmaye, E. (1992). Simulation-Based Language Learning: An Architecture and a Multimedia Authoring Tool. Technical Report 30, Institute for the Learning Sciences, Northwestern University.

Schank, R.C. (1991). Case-Based Teaching: Four Experiences in Educational Software Design. Technical Report 7, Institute for the Learning Sciences, Northwestern University.

7. Acknowledgments

We wish to acknowledge the members of the MOPed-II development team and users' group, including Laura Allender, Mark Chung, Scott Dooley, Jonathan Hickey, Jeff Lind, Marek Lugowski, Frank Luksa, Inna Mostovoy, Suzanne Pink and Wayne Schneider. MOPed-II developed out of an earlier authoring tool called MOPed, designed and built by Enio Ohmaye and Mark Chung. Kemi Jona designed and built the Teaching Executive and also contributed greatly to MOPed-II and several applications built with it. Ann Kolb, Jarrett Knyal, and Josh Tsui contributed excellent graphics work. Chip Cleary provided valuable comments on earlier drafts of this paper. We would also like to thank Roger Schank for his leadership.

This research was supported in part by the Advanced Research Projects Agency, monitored by the Office of Naval Research under grant no. N00014-91-J-4092. The Institute for the Learning Sciences was established in 1989 with the support of Andersen Consulting, part of The Arthur Andersen Worldwide Organization. The Institute receives additional support from Ameritech and North West Water Plc, Institute Partners.